不使用 Spring Cloud 如何实现 REST 服务调用

Spring Cloud Open Feign

REST 同步调用（Spring MVC 注解）

`@RequestMapping`

`@RequestParam`

不支持异步调用

```
@FeignClient("app-name")
public class HelloWorld {

    @RequestMapping("/hello-world")
    public String helloWorld(){
        return "Hello,World";
    }

}
```

- 负载均衡
- 熔断
- 重试

JAX-RS

注解类比

Spring Web MVC	JAX-RS	语义
@MatrixParam	`@MatrixVariable`	从 URI 获取 matrix 参数
@RequestParam	**@QueryParam**	
@RequestHeader	@HeaderParam	
	@CookieParam	
@PathVariable	@PathParam	
@ReqeustMapping	@Path	
@GetMapping	@GET	

Eclipse MicroProfile

实现步骤

实现 JAX-RS Client

编程特征

- JAX-RS 规范主要针对 REST 资源，面向 URI 编程
- 允许 HTTP 请求和响应进行序列和反序列，基于 POJO 编程，与传统 HTTP Client 不同（基于二进制流，InputStream 和 OutputStream）

实例说明

```
  Client client = ClientBuilder.newClient();
        Response response = client

.target("http://127.0.0.1:8080/hello/world")
   // WebTarget
                .request() //
Invocation.Builder
                .get();
            //  Response

        String content =
response.readEntity(String.class);
```

- 创建客户端 Client，通过 ClientBuilder#newClient()

实现步骤

- 通过 Java SPI 实现 javax.ws.rs.client.ClientBuilder 抽象类
- 实现 javax.ws.rs.client.Client 接口，并且将其通过 javax.ws.rs.client.ClientBuilder#build() 方法暴露
- 实现 javax.ws.rs.client.WebTarget
    - 实现 javax.ws.rs.core.UriBuilder
        - 语义：处理 URI
            - 设置 Path
            - 增加请求参数
            - 处理路径变量
        - 创建实例 javax.ws.rs.core.UriBuilder#newInstance()
            - 通过 Java SPI 实现 javax.ws.rs.ext.RuntimeDelegate
                - 实现 org.geektimes.rest.DefaultRuntimeDelegate#createUriBuilder()
    - 实现 javax.ws.rs.client.Invocation.Builder
        - 语义：HTTP 请求代理
        - 实现 HTTP 请求方法 Invocation 接口
            - 比如： HttpGetInvocation

- 实现序列化和反序列化

实现效果

- JAX-RS Client 帮助应用通过 URI 资源获取对应的 POJO

实现 MicroProfile Rest Client

编程特征

- MicroProfile Rest Client 规范在 JAX-RS 基础上，实现面向接口编程（类似于 OpenFeign）
- 允许 HTTP 请求和响应进行序列和反序列，基于 POJO 编程

案例说明

```
public class DefaultRestClientBuilderTest {

    @Test
    public void test() throws
MalformedURLException {
        HellWorld hellWorld =
RestClientBuilder.newBuilder()
                .baseUrl(new
URL("http://127.0.0.1:8080"))
                .build(HellWorld.class);

        // URI ->
http://127.0.0.1:8080/hello/world
```

```
 System.out.println(hellWorld.helloWorld());
    }
}

@Path("/hello")
interface HellWorld {

    @GET
    @Path("/world")
    String helloWorld();
}
```

实现步骤

- 实现 org.eclipse.microprofile.rest.client.spi.RestClientBuilderResolver SPI
- 实现 org.eclipse.microprofile.rest.client.RestClientBuilder 接口
- 实现目标接口的 JDK 动态代理
  - 对目标接口进行反射获取元信息（Metadata） - RequestTemplate
    - 获取接口和方法上的 @Path -> 相对路径信息
    - 确定方法上标注的那种 HTTP 方法注解，比如 @GET -> "GET"

    - 当获取完整 URI 之后，利用 JAX-RS Client 实现，发起 HTTP 请求 Response
    - 通过 Response 再反序列化成接口方法的返回类型
        - Response#readEntity(String.class)

学习规范技巧

- 与实际工作相结合
- Java 规范需要掌握 Java SPI - java.util.ServiceLoader